

Cloud Computing

Virtual Resources and Distributed Cloud Applications

Chapter 6: Cloud Platforms



Distributed and Operating Systems
Electrical Engineering and Computer Science
Technische Universität Berlin

Overview

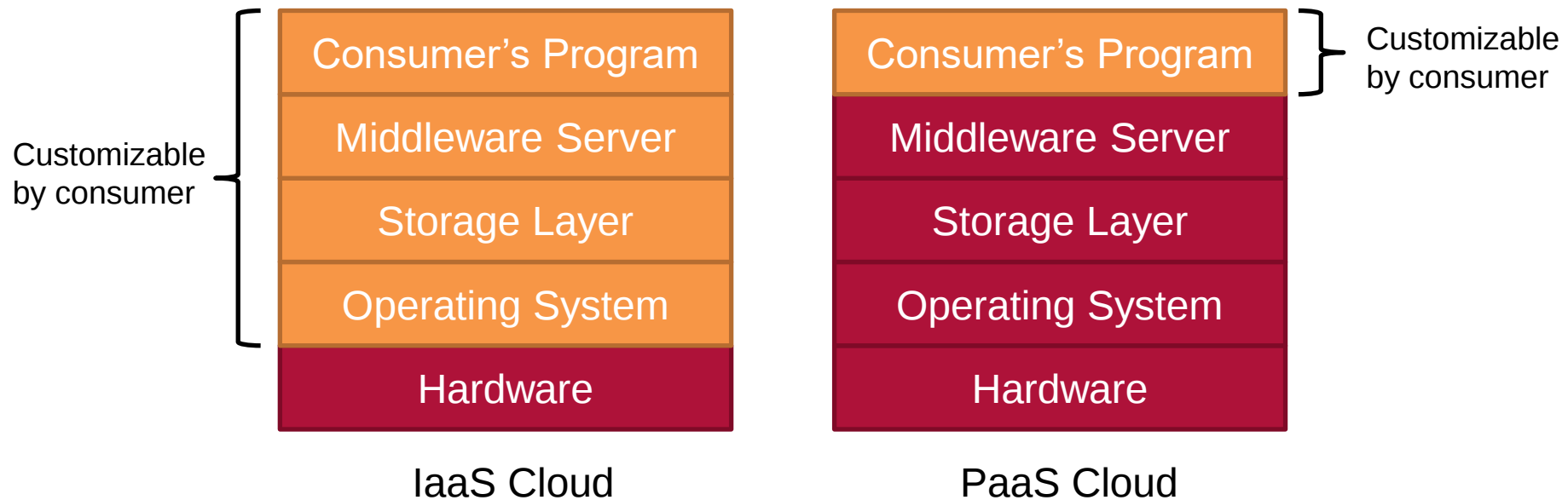
- Intro
 - **Platforms vs Infrastructure**
- Platforms
 - Azure
 - Amazon EMR and SageMaker
- Serverless Computing
 - Concept, FaaS, BaaS
 - Serverless Application Architectures
 - Example Platforms

Recap: PaaS Clouds

- Services offered by PaaS clouds (according to NIST_[2])
 - Programming languages
 - Libraries
 - Services
 - Tools
- Virtual images ready to deploy your software
- Characteristics of services
 - Consumer can deploy custom applications on PaaS cloud using the provider's application model
 - Consumer does not directly control the operating system, storage, and deployed runtime

IaaS vs. PaaS (1/2)

- PaaS offers higher abstraction level compared to IaaS
 - Less development/maintenance effort
 - Less flexibility, high provider dependence



IaaS vs. PaaS (2/2)

- Higher abstraction level enables other pricing models
- Service usage can be charged
 - by time
 - per query (e.g., for database services)
 - per message (e.g., for message queues)
 - per CPU usage (e.g., for request-triggered applications)
 - ...
- Enables interesting scenarios for consumers
 - Deployed applications can have very low operational costs until they become popular

PaaS Abstraction Levels

- Many levels of abstractions in PaaS
 - Execution environments (like on Heroku)
 - Databases (like DynamoDB)
 - Distributed processing (like EMR)
 - Domain-specific workbenches (like SageMaker)
- Complete services (like Rekognition)*
- A lot of new offerings are quite high-level PaaS

*almost SaaS, but mostly used by developers

PaaS Value Proposition

- Maintenance
 - Hardware maintenance
 - OS patches
 - Middleware updates
- Availability
 - Application/service will be available despite maintenance/outages
- Scalability
 - Application/service will scale to thousands of concurrent users

Overview

- Intro
 - Cost of Scalability
 - Platforms vs Infrastructure
- Platforms
 - **Azure**
 - Amazon EMR and SageMaker
- Serverless Computing
 - Concept, FaaS, BaaS
 - Serverless Application Architectures
 - FaaS Platform Examples
 - Stateful Functions

Microsoft Azure

- Microsoft's cloud computing platform
- Launched in 2010 as PaaS solution
 - Targeted at scalable, reliable web applications
- Initially, platform was composed of three components
 - Microsoft Azure: Compute and storage services
 - SQL Azure: Cloud-based DBMS
 - Azure AppFabric: Tools to bridge gap between local and cloud-hosted applications



Geographic Distribution of Azure Data Centers



- Microsoft calls each geographic location a region
 - Within a region, customers can create affinity groups to deploy services as closely together as possible

Microsoft Azure Affinity Groups

- Some data centers are built from shipping containers
 - Each container can contain up to 2500 servers



Images
from [3]

- Affinity group: Logical grouping of components
 - Azure tries to deploy components as closely as possible
 - For example within the same container, if possible

Microsoft Azure Affinity Groups

- Project Natick tries to go further in global distribution with underwater data centers

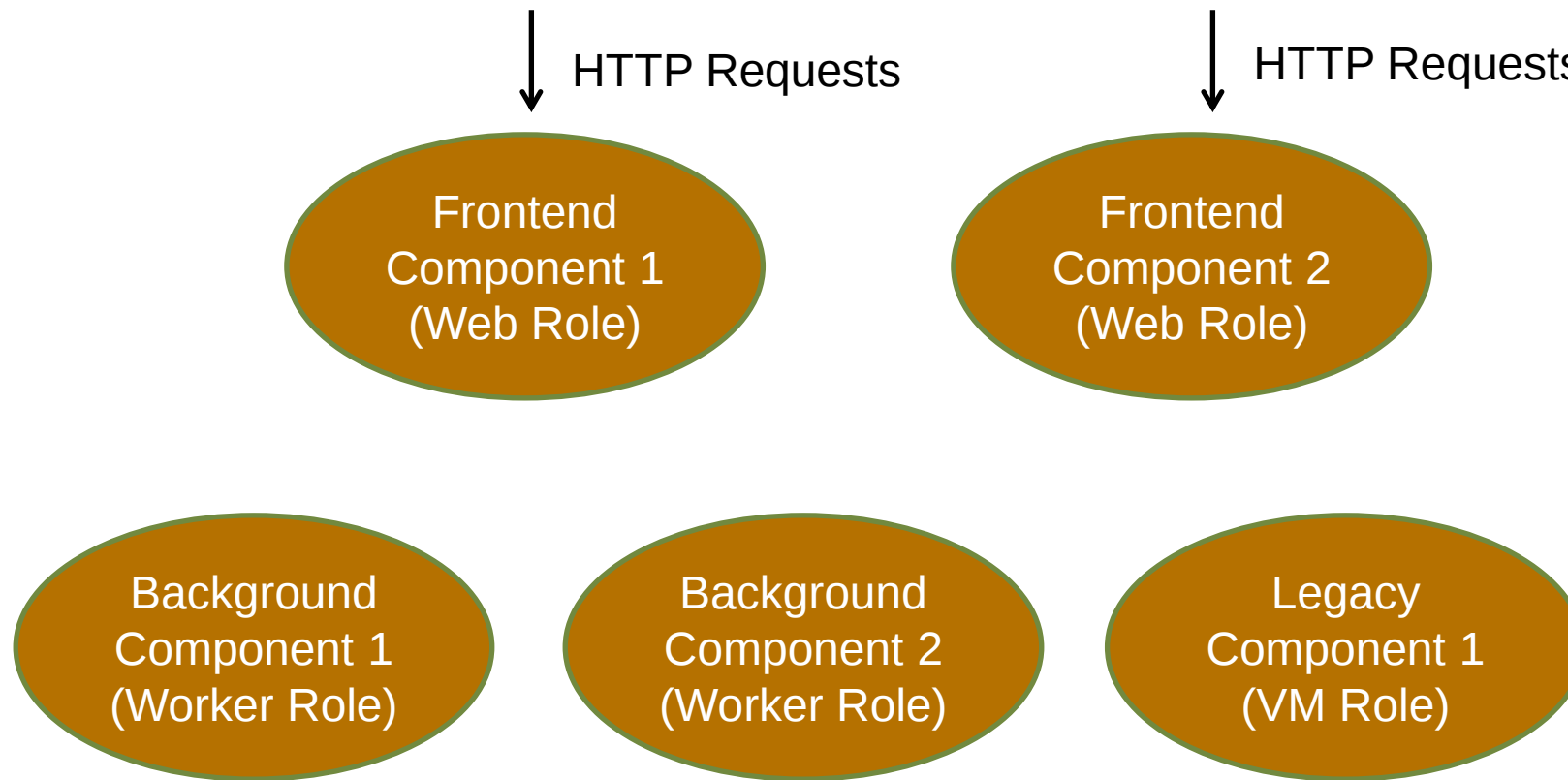


Microsoft Azure Programming Model (1/4)

- Azure applications separated into logical components
 - Each component is assigned to a role
 - Multiple instances of the same component might exist
- Azure programming model offers three roles
 - Web role: Components facing the outer world
 - Accepting requests via HTTP, e.g. Web sites/services
 - Worker role: Components doing background tasks
 - VM role: Legacy components
 - Component outside of Azure's programming model

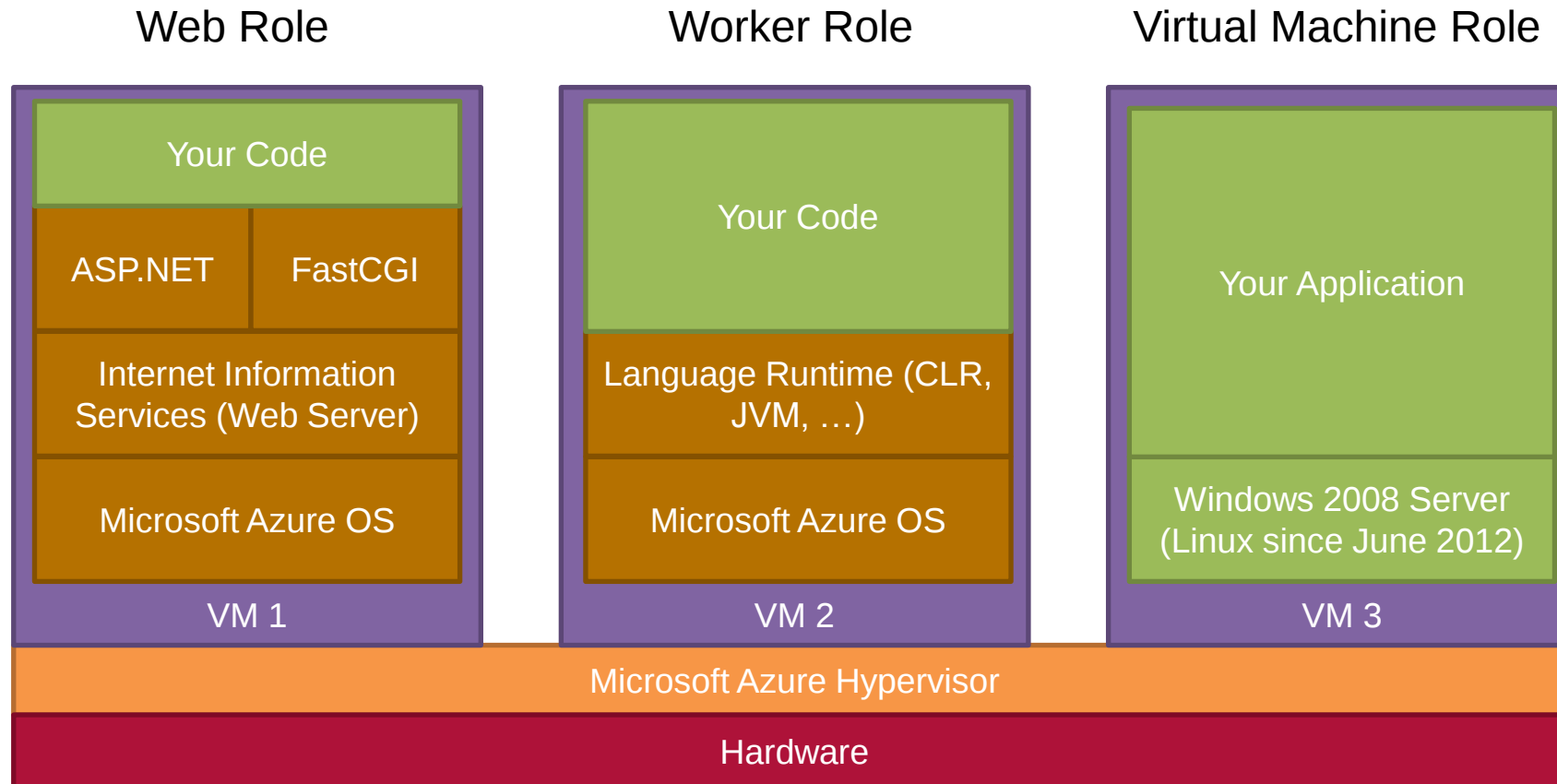
Microsoft Azure Programming Model (2/4)

- Example of a Microsoft Azure application according to the programming model



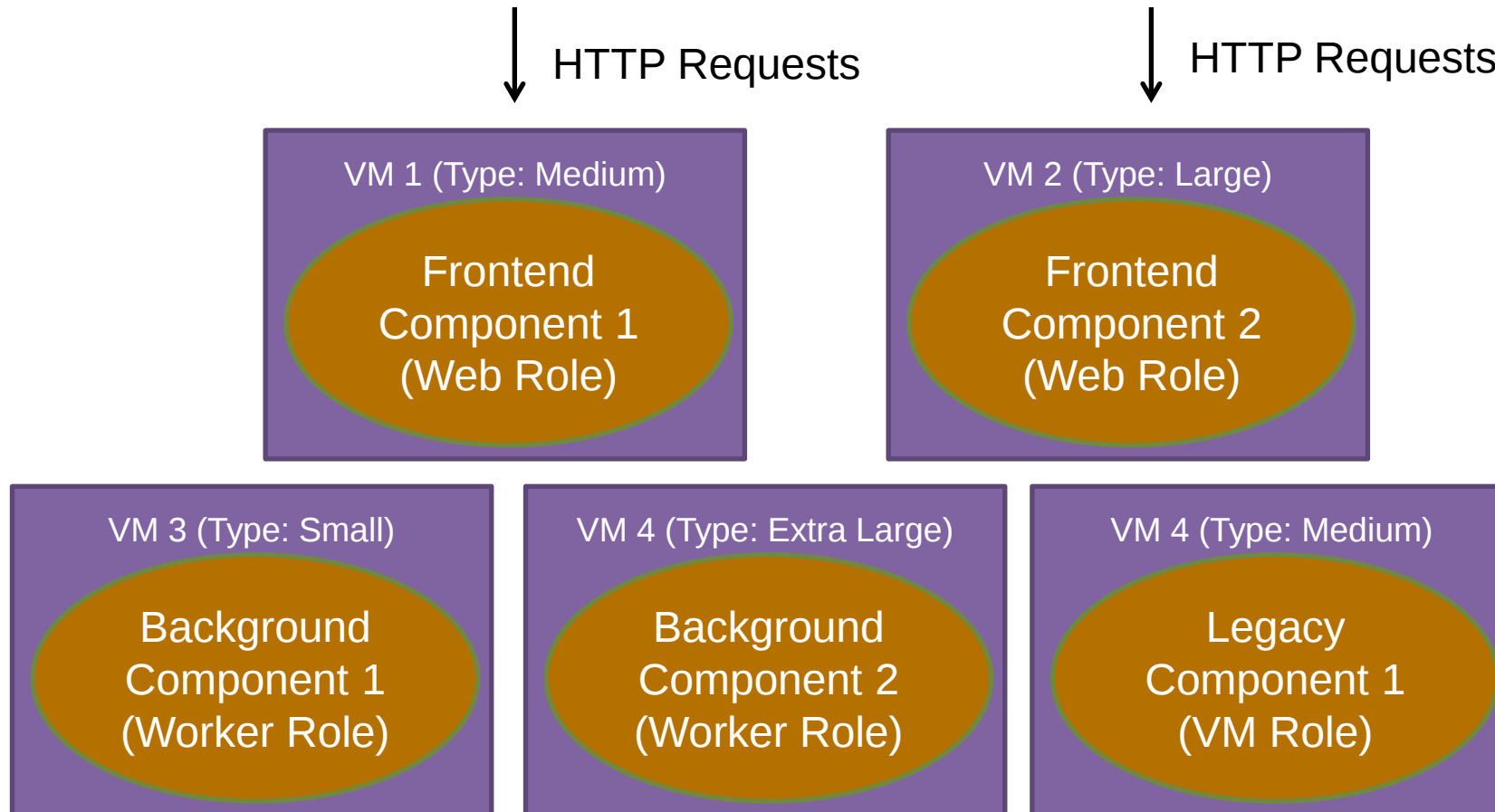
Microsoft Azure Programming Model (3/4)

- Different roles offer different levels of abstraction



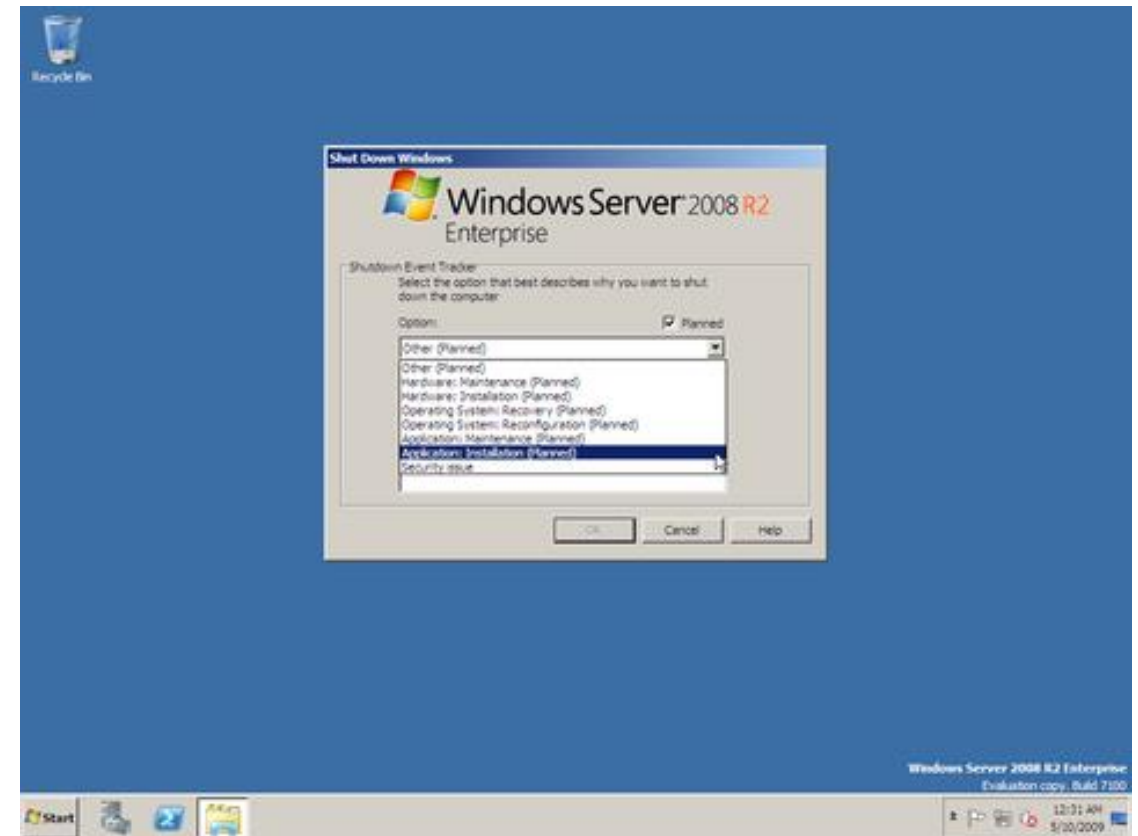
Microsoft Azure Programming Model (4/4)

- Example application with virtual machine isolation



Entry Points into Microsoft Azure Components (1/3)

- VM Role entry point: Precompiled VM image
- Azure is agnostic of guest OS
 - No automatic patches or updates
 - Basically IaaS abstraction
 - Intended for legacy components only



Entry Points into Microsoft Azure Components (2/3)

- Worker Role entry point: Archive with intermediate code
 - Code must implement specific interface

```
namespace WorkerRole1
{
    public class WorkerRole : RoleEntryPoint
    {
        public override void Run()
        {
            // Code to be executed during role's life time
        }
    }
}
```

Entry Points into Microsoft Azure Components

(3/3)

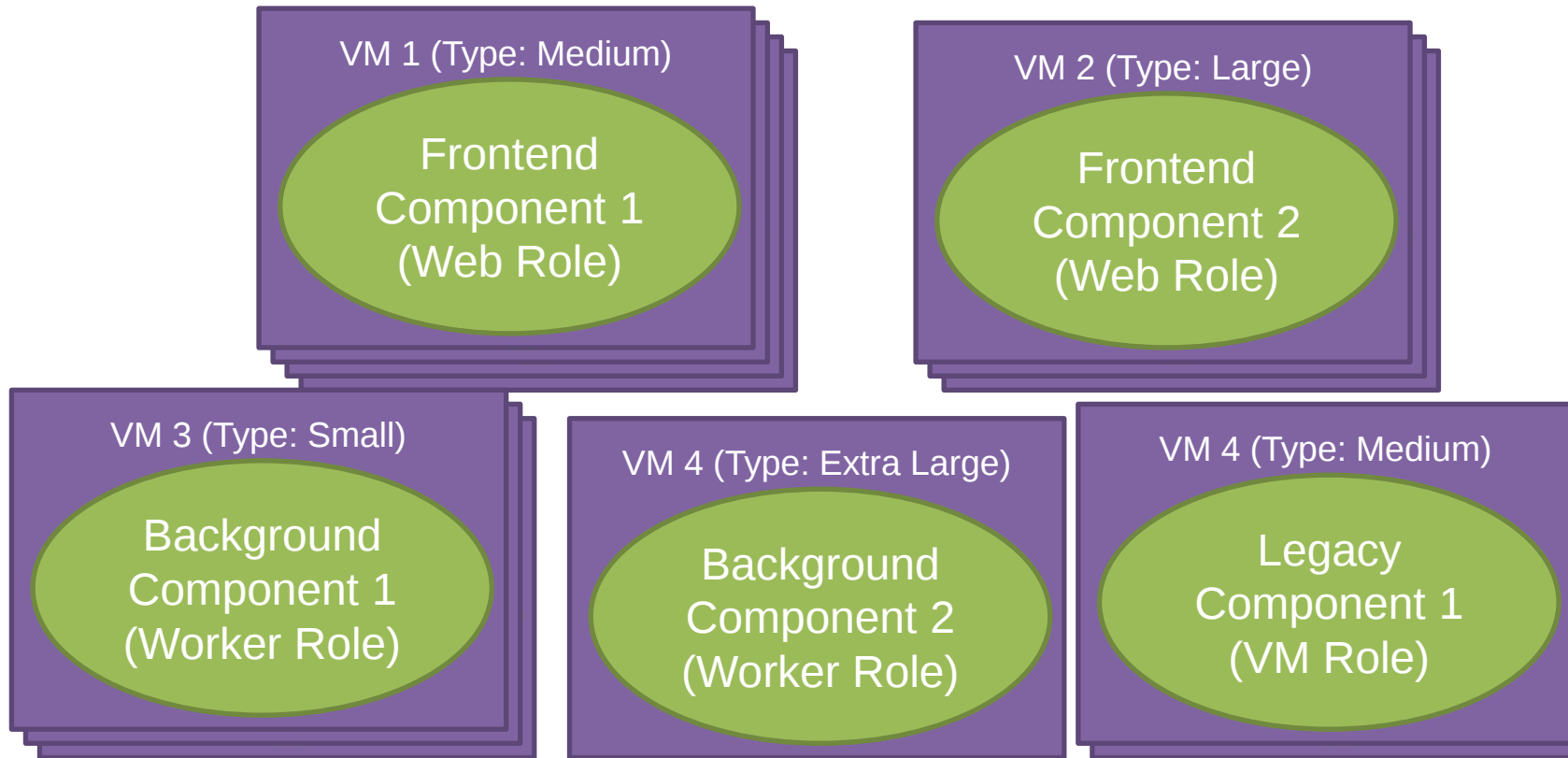
- Web Role entry point: Archive with web code
 - Either ASP.NET code or code compliant with FastCGI

```
@{
    // Some .NET code to be evaluated when page is rendered
    ViewBag.Title = "About Us";
}

<h2>Your heading</h2>
<p>
    The content of your website.
</p>
```

How To Achieve Scalability?

- Azure's answer: Run many instances of each role



Microsoft Azure Platform Maintenance

- How to deal with OS patches, middleware updates, ...?
- Azure's approach
 1. Bring up new instance of role on new HW / patched OS image
 2. Redeploy customer's code
 3. Register new instance with the load balancer
 4. Kill outdated instance
 5. Continue until all old instances have been replaced
- Prerequisite: OS and user code can be separated (not true for VM Role)

Overview

- Intro
 - Cost of Scalability
 - Platforms vs Infrastructure
- Platforms
 - Azure
 - **Amazon EMR and SageMaker**
- Serverless Computing
 - Concept, FaaS, BaaS
 - Serverless Application Architectures
 - FaaS Platform Examples

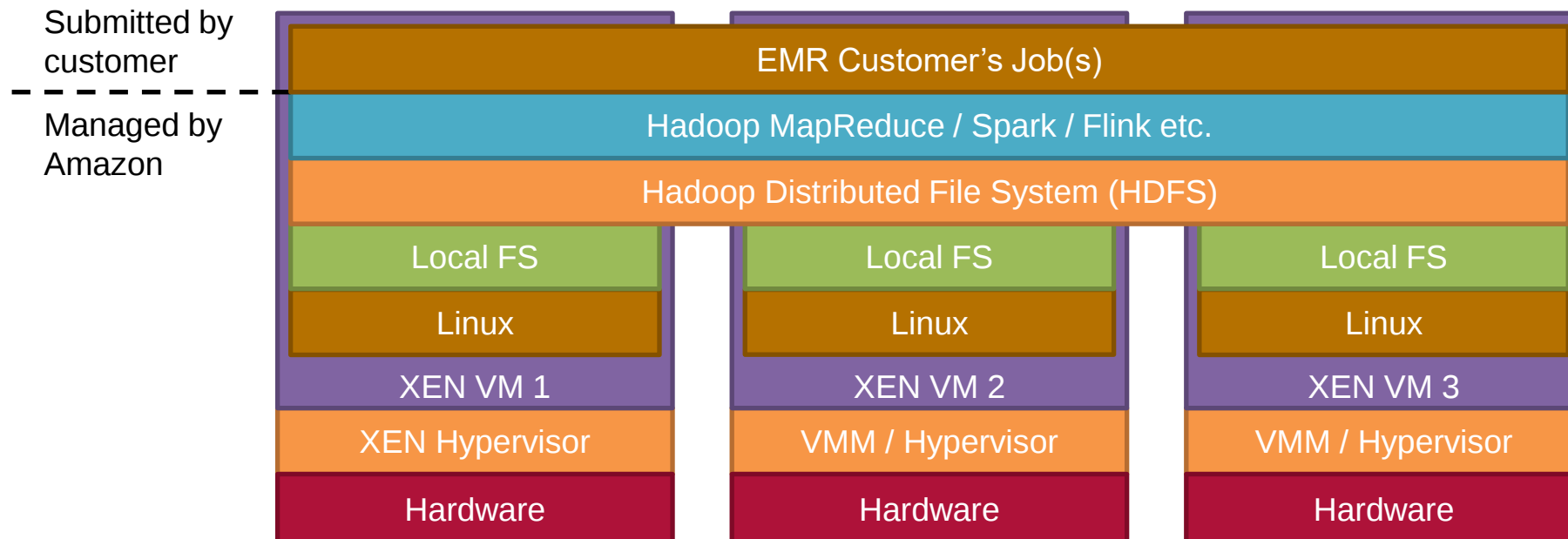
Amazon EMR

- Cloud service for data-intensive applications
 - Introduced in 2009 as part of AWS
- Started with Hadoop MapReduce on EC2 and integrated with S3 storage
 - No setup/configuration effort for the EC2 resources
 - Follows pay-per-use model (billed by the second)
- Supports many processing systems (e.g. Spark and Flink) and storage services (e.g. Amazon DynamoDB) today



Software Stack of Amazon EMR

- Amazon runs preconfigured systems on EC2
 - User submits/monitors jobs through web interface
 - Dedicated VMs per user
 - No direct access to the VMs (in contrast to IaaS)



Job Processing Cycle on Amazon EMR (1/2)

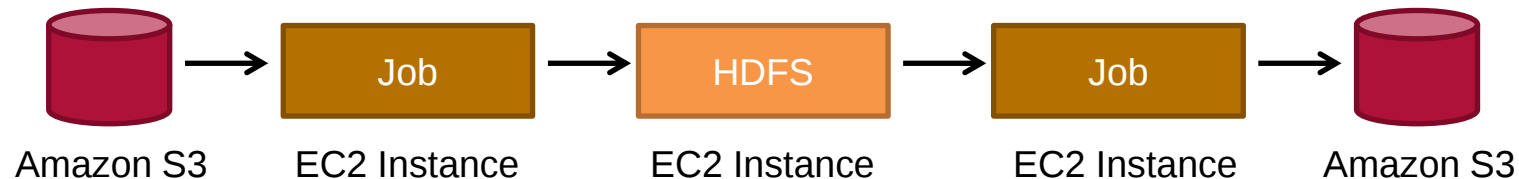
1. Customer submits job through web interface
 - Job specification contains
 - ◆ Location of input data on Amazon S3
 - ◆ Framework, job code, user libraries, parameters, ...
 - ◆ Number of virtual machines to run the job on
 - ◆ Type of virtual machines to run the job on
 - ◆ Designated output location on Amazon S3
2. Requested virtual machines are started on EC2
 - Pricing starts to apply
 - Boot of EC2 instances with preconfigured systems and start of these
 - Worker nodes automatically contact master

Job Processing Cycle on Amazon EMR (2/2)

3. Job runs on rented VMs
 - Input read from Amazon S3
 - Customer can monitor execution progress through web interface
 - Output saved to Amazon S3
4. After job completion, EC2 instances are automatically shut down
 - Job completed after all distributed tasks have finished
 - All VMs must remain allocated until that point in time
 - ◆ Intermediate data might be required for potential recovery
 - Shutdown also marks end of billing period

Executing Sequences of Jobs

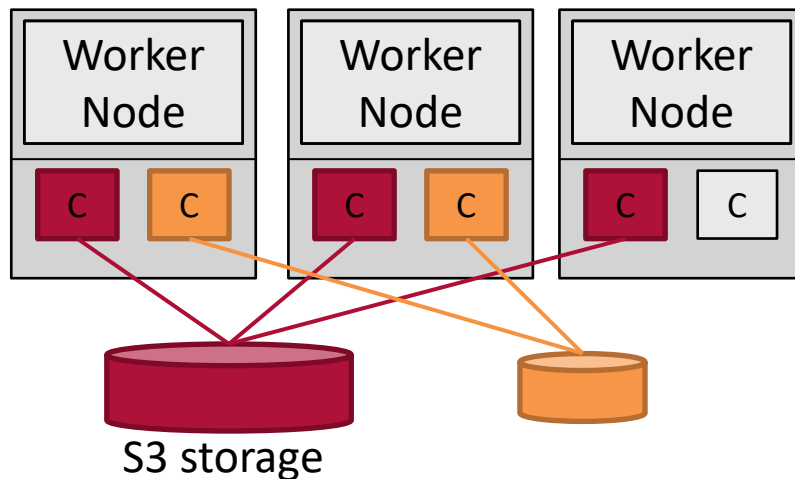
- Deployment model requires to store initial input/final output on Amazon S3
 - Violates principle to keep data local to computation, so increased effort to move data to/from virtual machines
 - Yet storage nodes can be independently scaled
- EMR also allows to execute sequences of jobs
 - Initial input/final output still stored on Amazon S3
 - Intermediate results between jobs stored in HDFS
 - ◆ Data kept at least local between two, e.g., MR jobs



Data Stored in S3

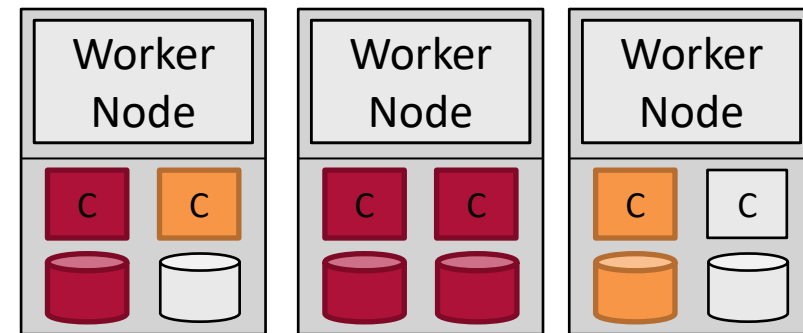
- Data stored in S3 in-between jobs
 - Pro: independent number of storage/compute nodes and less expensive than a permanent HDFS cluster
 - Contra: data locality
 - Thus, good if you only run jobs every now and then

EMR setup

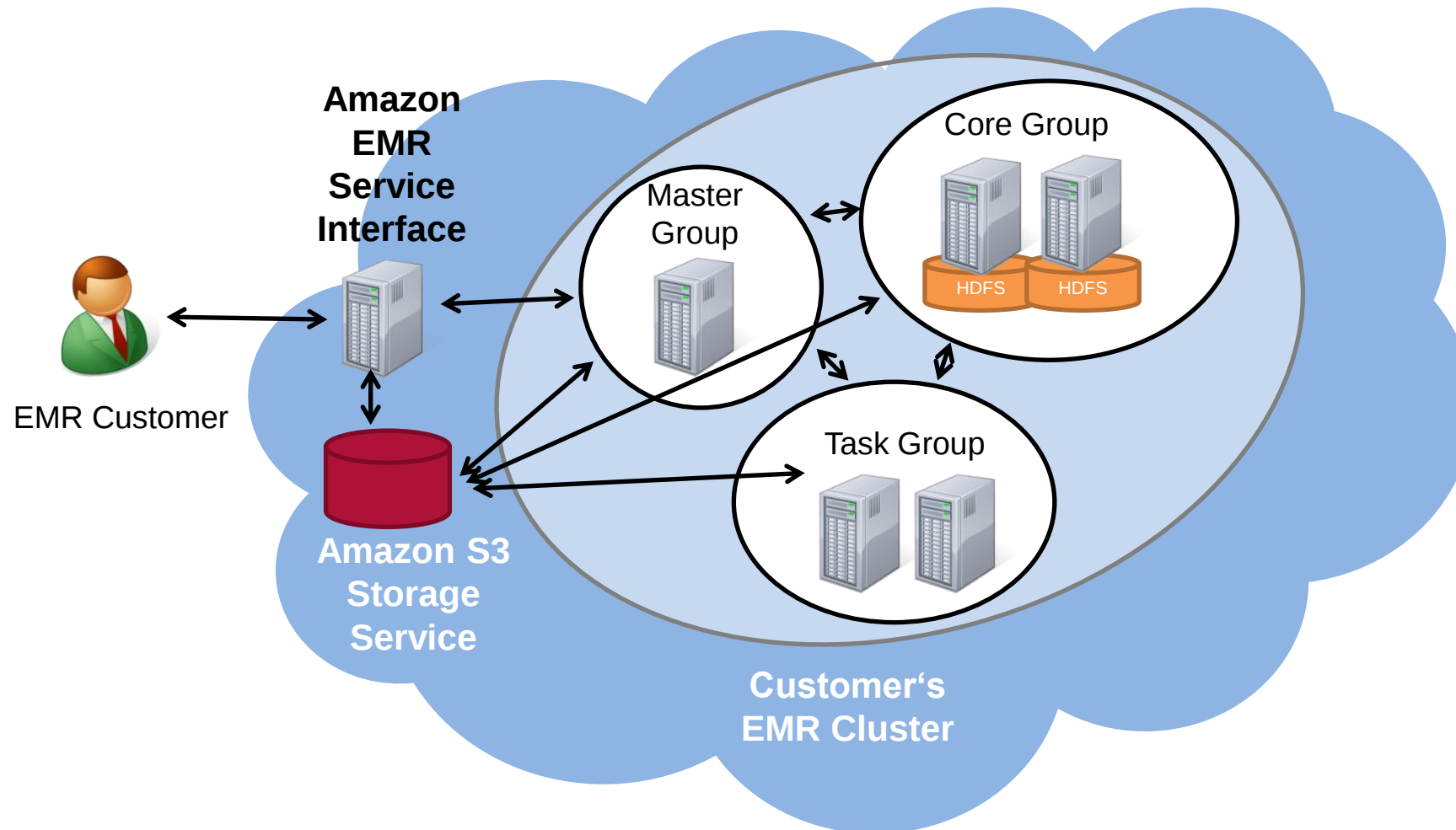


VS.

Dedicated analytics cluster



EMR Architectural Summary



Auto-Scaling with EMR

- EMR allows you to set rules for dynamically scaling of core and task nodes based on metrics
 - e.g., scale-up number of workers when less than 15% of memory is available for a five-minute period

The image shows a screenshot of the AWS EMR console's auto-scaling rule configuration interface. The rule is named "MyScalingRule". It is configured to add 1 instance when the "YARNMemoryAvailablePercentage" CloudWatch metric is less than or equal to 15 percent for 1 five-minute period. The cooldown period is set to 1 five-minute period. Callouts identify the following components: Rule name, Scaling adjustment, CloudWatch metric, Evaluation period, Comparison operator, Threshold, and Cooldown.

Rule name	MyScalingRule
Add	1 Instances
if	YARNMemoryAvailablePercentage
is	less than or equal to
	15 percent
for	1 five-minute periods
Cooldown period	1 five-minute periods

<https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-automatic-scaling.html>

Amazon SageMaker

- Domain-specific service for machine learning tasks, started in late 2017
- Facilitates the steps of a machine learning workflow
 - Labelling data
 - Building
 - Training
 - Deployment

Amazon SageMaker: Labelling Data

- Data labelling by humans
- Mechanical workers via amazon or self-recruited
- Common labelling types have a predefined task type
- Active learning for faster labelling

Task type [Info](#)

Task selection
Select the task that a human worker will perform to label objects in your dataset.

☒ **Image classification**
Get workers to categorize images into specific classes. [Info](#)

☒ Basketball
☐ Soccer



☐ **Bounding box**
Get workers to draw bounding boxes around specified objects in your images. [Info](#)



☐ **Text classification**
Get workers to categorize text into specific classes. [Info](#)

☒ Positive
☐ Negative

'The movie tells a lovely and wise story with honesty and has been acted out with unassuming grace.'

☐ **Semantic segmentation**
Get workers to draw pixel level labels around specific objects and segments in your images. [Info](#)



☐ Custom

Predefined task types for
labelling images

Amazon SageMaker: Labelling Data Process (1/2)

- Data is uploaded to s3
- Labelling Job is submitted, consisting of
 - Instructions to human workers
 - Reference to unlabeled data
 - Definition of Labels, if fixed
 - Email addresses of workers, if self-recruited

no-reply@verificationemail.com

You're invited by undefined to work on a labeling project.

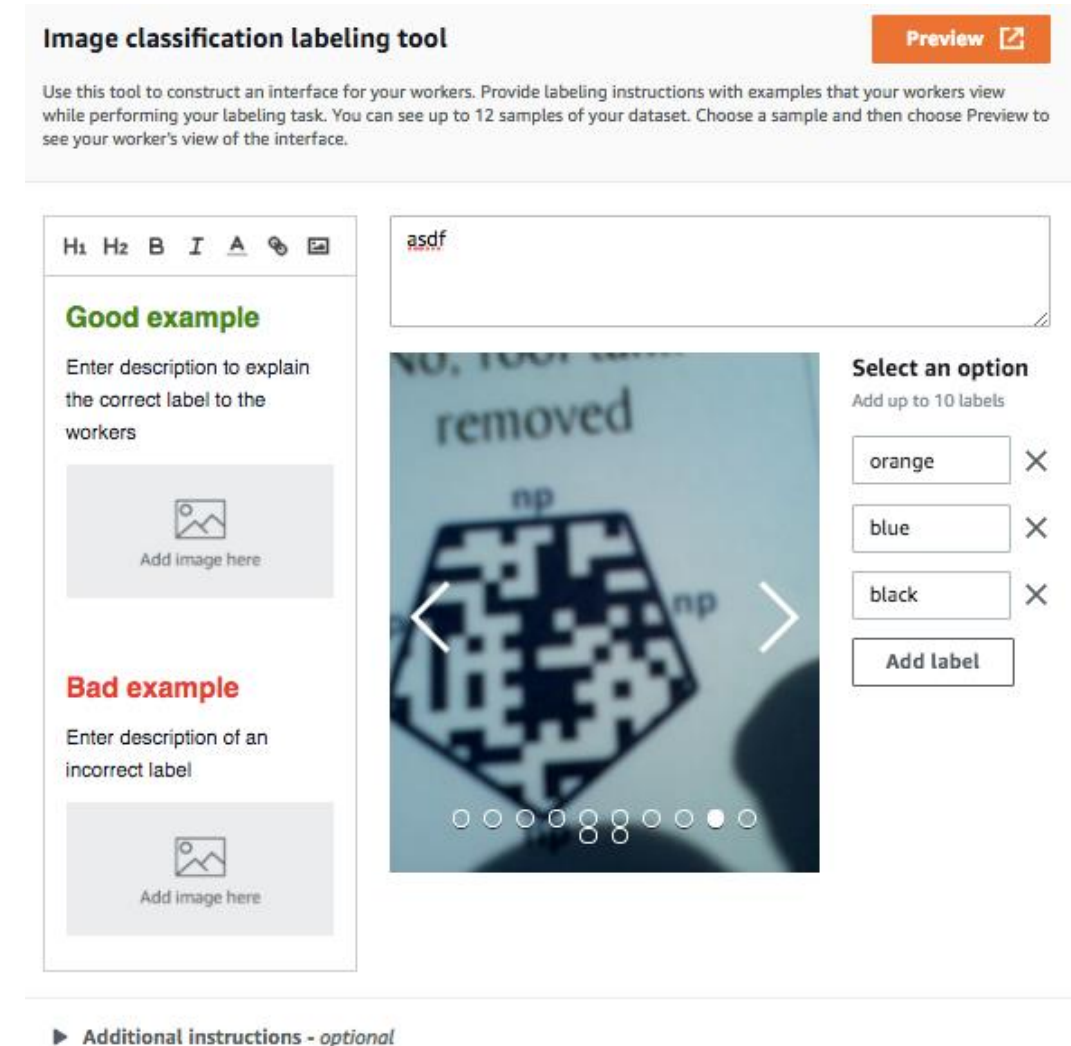
To:



You're invited to work on a labeling project.

Amazon SageMaker: Labelling Data Process (2/2)

3. Human workers label data via web interface
4. Labelling can be accelerated via active learning
 - Active learning used to automatically select which data need to be labelled manually



Amazon SageMaker: Building and Training Jobs

- Options to build models:
 - Often Jupyter Notebooks
 - Simpler problems: via CLI, JSON, REST
- There are many built-in algorithms for tasks like image classification, text classification, and clustering

```
In [ ]: # create the Amazon SageMaker training job
sagemaker = boto3.client(service_name='sagemaker')
sagemaker.create_training_job(**training_params)

# confirm that the training job has started
status = sagemaker.describe_training_job(TrainingJobName=job_name)['TrainingJobStatus']
print('Training job current status: {}'.format(status))

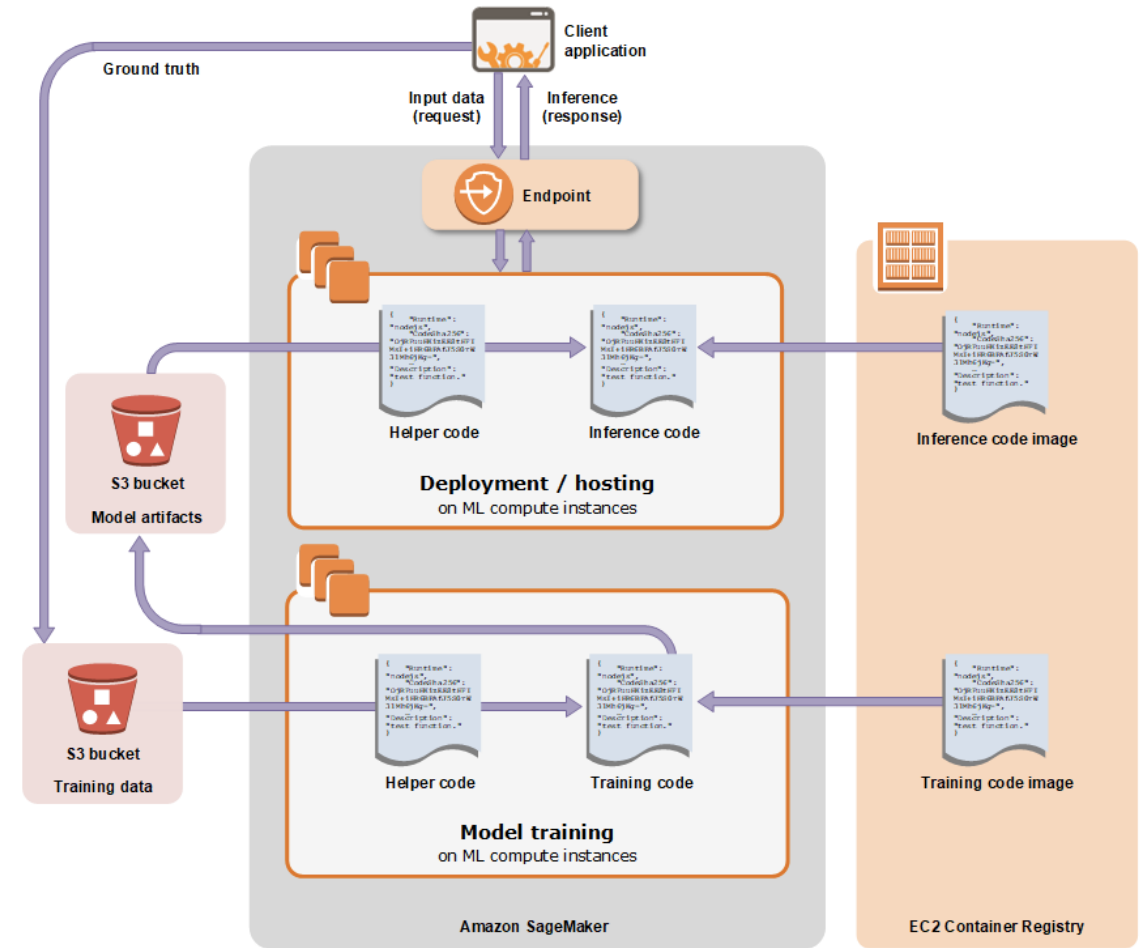
try:
    # wait for the job to finish and report the ending status
    sagemaker.get_waiter('training_job_completed_or_stopped').wait(TrainingJobName=job_name)
    training_info = sagemaker.describe_training_job(TrainingJobName=job_name)
    status = training_info['TrainingJobStatus']
    print("Training job ended with status: " + status)
```

Amazon SageMaker: Building and Training Jobs

- SageMaker integrates popular ML-libraries like MXNet, Tensorflow, and PyTorch
- Alternatively, customers can build docker containers to
 - Use specific unsupported versions
 - Use a ML-framework that's not provided by SageMaker
 - Use unsupported additions to a framework

Amazon SageMaker: Deployment

- Trained models can be deployed as
 - Web endpoints
 - Batch transform
- Autoscaling deployed models based on
 - Target metric
 - Scaling policy
 - Integrated with CloudWatch



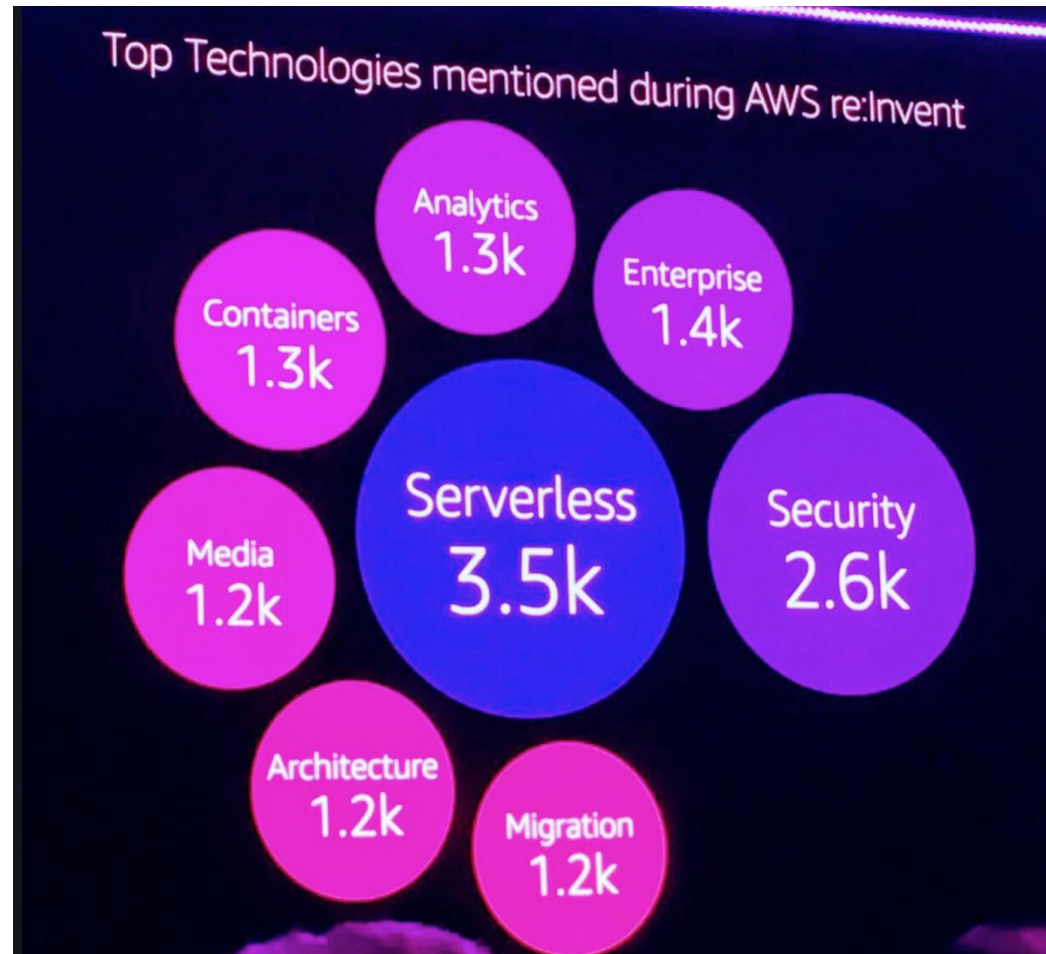
Overview

- Intro
 - Cost of Scalability
 - Platforms vs Infrastructure
- Platforms
 - Azure
 - Amazon EMR and SageMaker
- **Serverless Computing**
 - Concept, FaaS, BaaS
 - Serverless Application Architectures
 - FaaS Platform Examples

Serverless Computing^[11]

- A cloud execution model
 - Provider runs the server-side and dynamically manages resources
- Concept for the logical next step of abstraction
- Picked up speed since around 2015
 - Marketed by all major cloud providers
 - Open source projects to facilitate serverless execution

AWS re:Invent 2018



Overview

- Intro
 - Cost of Scalability
 - Platforms vs Infrastructure
- Platforms
 - Azure
 - Amazon EMR and SageMaker
- Serverless Computing
 - **Concept, FaaS, BaaS**
 - Serverless Application Architectures
 - FaaS Platform Examples

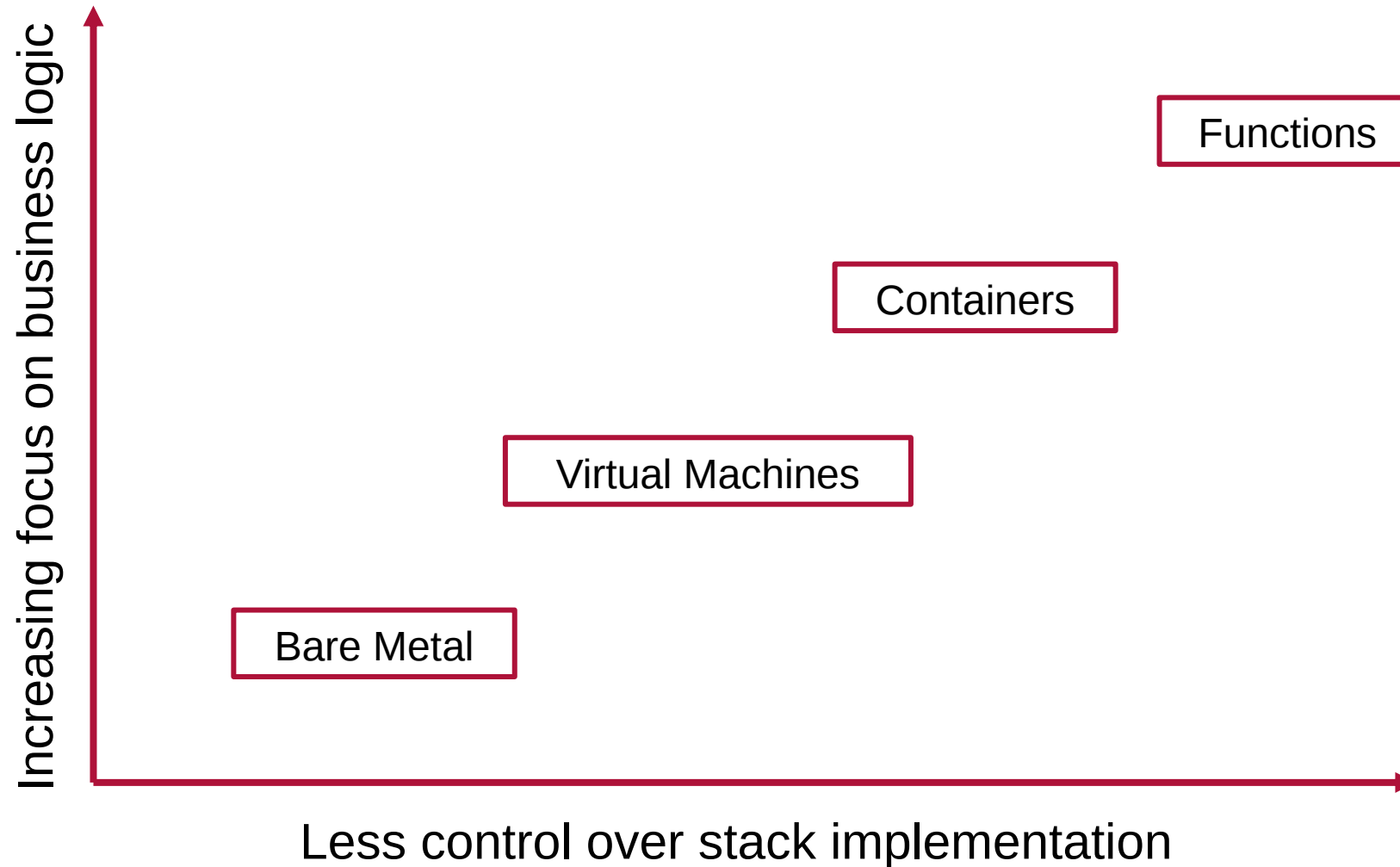
Serverless and PaaS

- Serverless Computing is an execution model that is enabled by Platform as a Service offerings
- Serverless uses
 - Platform services that do not live long
 - Platform services where you do not think about scaling
 - Serverless scaling is automatically managed, transparent, and fine-grained

Serverless: FaaS and BaaS

- Enabled by high-level PaaS offerings:
 - Backend as a Service
 - ◆ Use cloud database, authentication service, etc. as backend to a rich client app
 - ◆ Often used for mobile apps
 - ◆ Examples: Firebase, Parse Platform, AWS Cognito
 - Function as a Service
 - ◆ Execute own code in event-triggered, and ephemeral, and stateless compute containers
 - ◆ Examples: AWS Lambda, Google Cloud Functions, Azure Functions

Serverless Computing

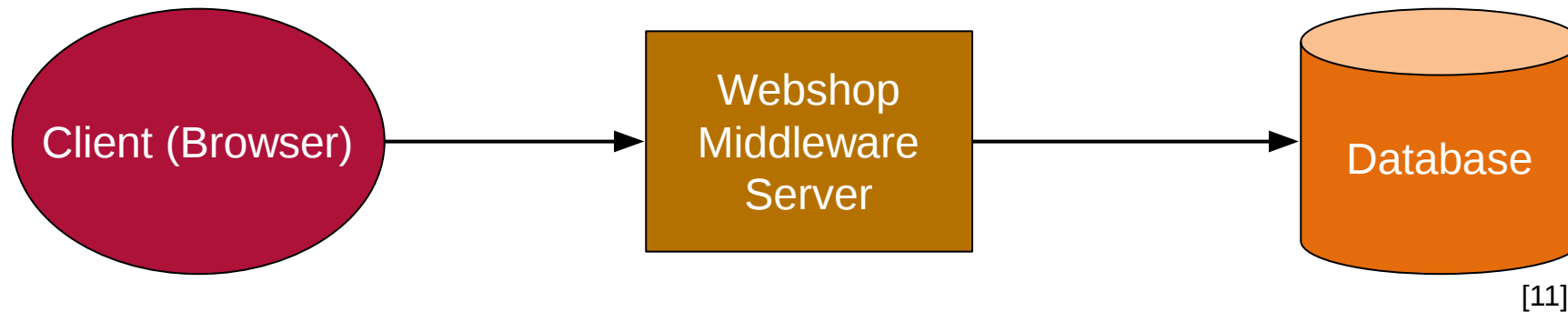


Overview

- Intro
 - Cost of Scalability
 - Platforms vs Infrastructure
- Platforms
 - Azure
 - Amazon EMR and SageMaker
- Serverless Computing
 - Concept, FaaS, BaaS
 - **Serverless Application Architectures**
 - FaaS Platform Examples

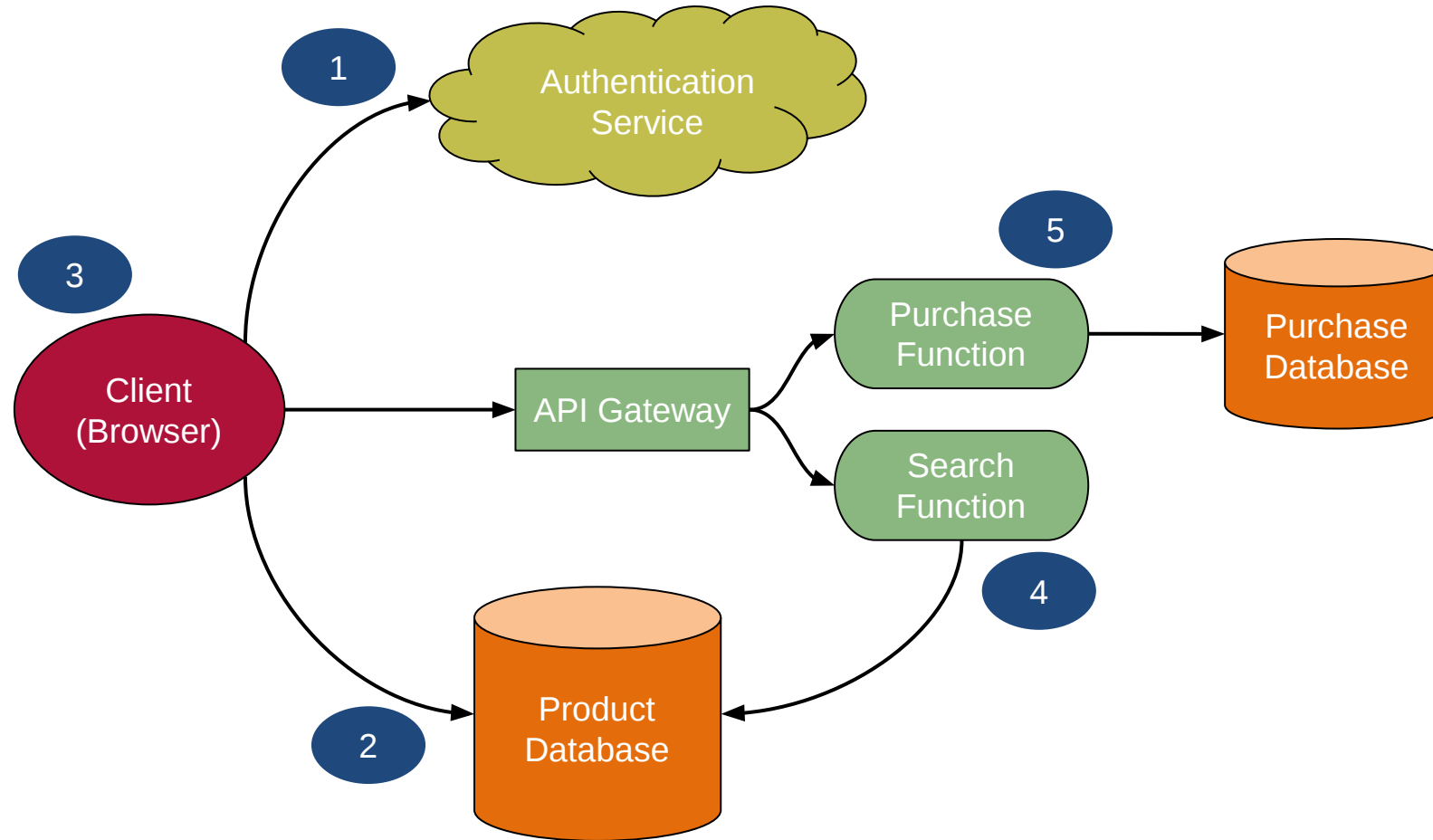
Serverless Architectures: Illustrating Examples (1/3)

- A webshop as a traditional three-tier web application:



Serverless Architectures: Illustrating Examples (2/3)

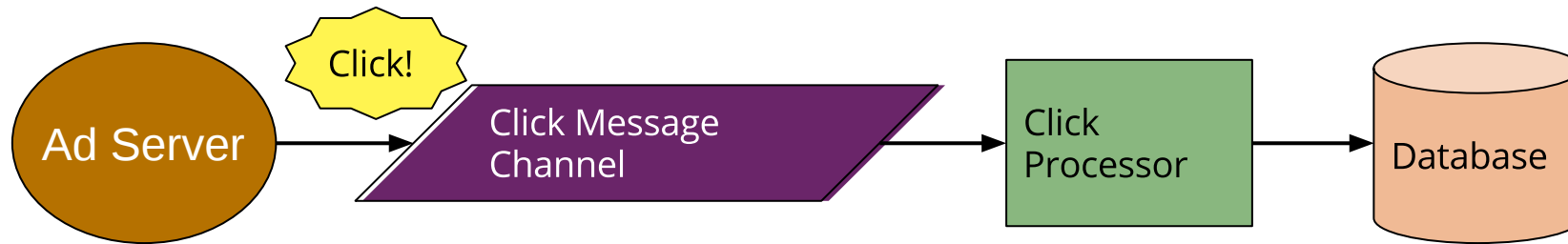
- A possible serverless architecture for the webshop:



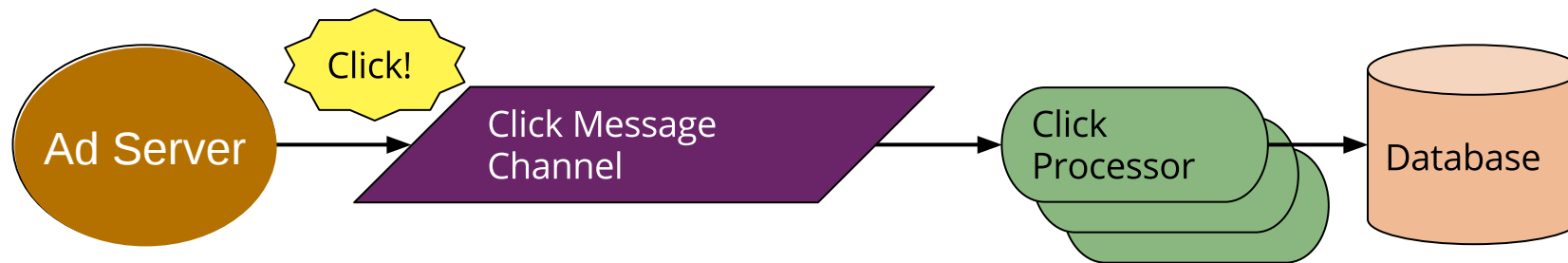
[11]

Serverless Architectures: Illustrating Examples (3/3)

- Tracking clicks as an example of backend data-processing:



- A possible serverless architecture for the service:



Function as a Service

- Motivation: Run backend code without managing server resources and/or long-running applications
- Architecture: Event-triggered, ephemeral, stateless
- Automatic, transparent, and fine-grained horizontal scaling
- Deployment is uploading code and artifacts (like it is for many PaaS services)
- No restrictions regarding languages and frameworks

Function Triggers

- Functions are triggered by events
 - File changes in file storage (like S3)
 - Messages from a messaging system (like Kafka)
 - Database events
 - Scheduling / scheduled tasks
 - Requests to an HTTP endpoint (possibly via an API gateway)

Function as a Service: Restrictions

- State
 - FaaS often described as stateless
 - While state can be kept in execution container, there are no guarantees when container will disappear
 - Persist state by externalizing it (e.g., write to a database or a file service)
- Timing
 - Restricted execution duration of functions (e.g., 15 mins)
 - Forces a clear separation of coordination and execution
 - Startup latencies vary (milliseconds to seconds)

API Gateways

- BaaS service to offer (usually RESTful) APIs by configuring a mapping between routes and endpoints
- Offer many default tasks for API endpoints
 - Authorization
 - Rate limiting
 - Monitoring
 - Version management
- Often used in conjunction with FaaS, routing incoming requests to specific functions
- Well-suited for microservice architectures

Serverless Architecture Implications (1/2)

- Less Ops to worry about at the expense of flexibility
- Increasing vendor lock-in as with all higher PaaS abstractions
- Coordination vs. execution split is forced
- Change of structure of cloud expenses
 - From paying for resources reserved upfront to paying for capacities actually used
 - Increased compartmentalization and modularization in production is possible

Serverless Architecture Implications (2/2)

- Concentrate on the code that creates real business value, enabling a lean product development
- Simpler experimentation
 - Cloud services implement version management
 - API Gateway can redirect requests to named versions

When to use Serverless Architectures

- Good:
 - Less application code, reduced development effort
 - Cloud operator probably better at implementing default tasks than application developer
 - Possibly less costs
- Bad:
 - Additional complexity through managing the different cloud services
 - Monitoring can become more complicated
 - Security needs to be handled at all the services

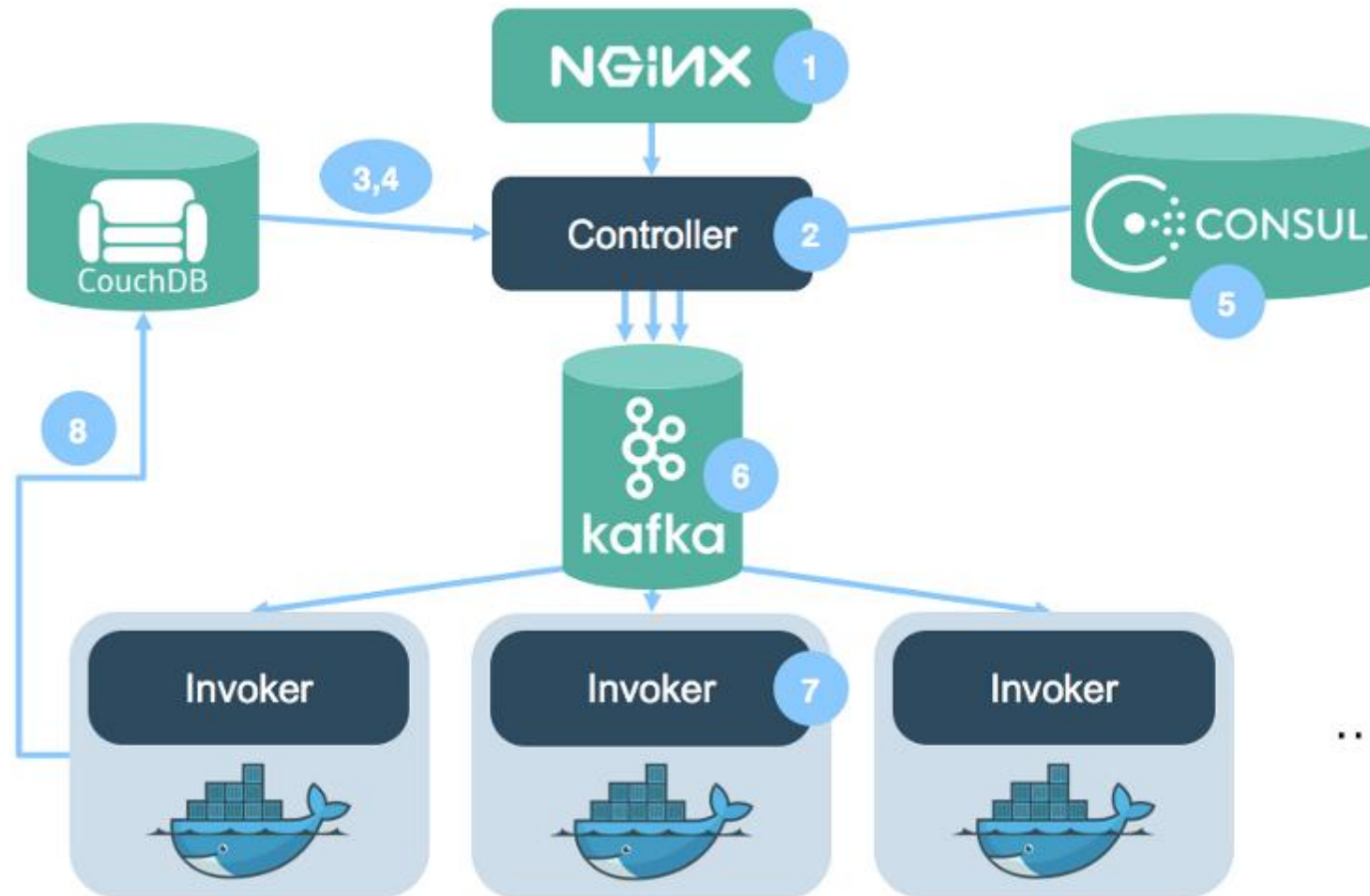
Overview

- Intro
 - Cost of Scalability
 - Platforms vs Infrastructure
- Platforms
 - Azure
 - Amazon EMR and SageMaker
- Serverless Computing
 - Concept, FaaS, BaaS
 - Serverless Application Architectures
 - **FaaS Platform Examples**

Serverless Landscape

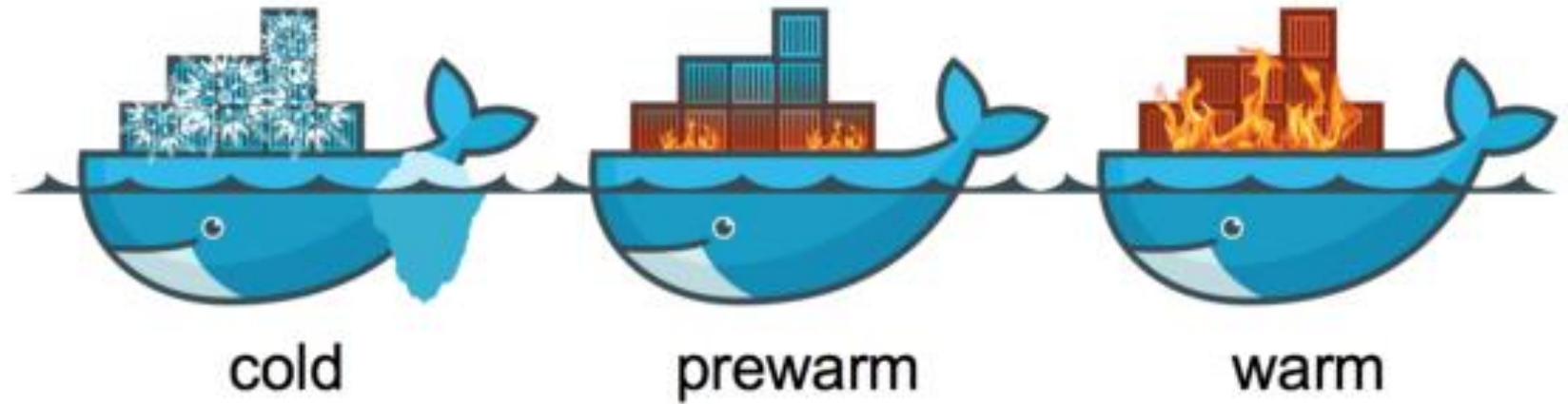


Apache OpenWhisk



[13]

Apache OpenWhisk Invoker



Starting the container

Initializing the action

Running the action



FaaS Performance Comparison^{[9][10]}

Peeking Behind the Curtains of Serverless Platforms

Liang Wang¹, Mengyuan Li², Yinqian Zhang², Thomas Ristenpart³, Michael Swift¹

¹UW-Madison, ²Ohio State University, ³Cornell Tech

Abstract

Serverless computing is an emerging paradigm in which an application's resource provisioning and scaling are managed by third-party services. Examples include AWS Lambda, Azure Functions, and Google Cloud Functions. Behind these services' easy-to-use APIs are opaque, complex infrastructure and management ecosystems. Taking on the viewpoint of a serverless customer, we conduct the largest measurement study to date, launching more than 50,000 function instances

Serverless computing originated as a design pattern for handling low duty-cycle workloads, such as processing in response to infrequent changes to files stored on the cloud. Now it is used as a simple programming model for a variety of applications [14, 22, 42]. Hiding resource management from tenants enables this programming model, but the resulting opacity hinders adoption for many potential users, who have expressed concerns about: security in terms of the quality of isolation, DDoS resistance, and more [23, 35, 37, 40]; the need to understand resource management to improve application

Evaluation of Production Serverless Computing Environments

Hyungro Lee, Kumar Satyam and Geoffrey C. Fox
School of Informatics, Computing and Engineering
Indiana University Bloomington
{lee212, ksatyam, gcf}@indiana.edu

Abstract—Serverless computing provides a small runtime container to execute lines of codes without infrastructure management which is similar to Platform as a Service (PaaS) but a functional level. Amazon started the event-driven compute named Lambda functions in 2014 with a 25 concurrent limitation, but it now supports at least a thousand of concurrent invocation to process event messages generated by resources like databases, storage and system logs. Other providers, i.e., Google, Microsoft, and IBM offer a dynamic scaling manager to handle parallel requests of stateless functions in which additional containers are provisioning

Microsoft Azure already have the per-second billing, but it even costs every second whether a program runs or not.

Serverless is a miss-leading terminology because it runs on a physical server but it succeeded in emphasizing no infrastructure configuration along with the preparation of computing environments. Fox et al [1] defines serverless computing among other existing solutions, such as Function-as-a-Service (FaaS) and Event-Driven Computing, and we see produc-

Limits of FaaS_[14,15]

- Current limitations (almost by definition):
 - No long-lived functions
 - No efficient access to persistent state / data
 - No composition of and efficient communication between functions that make up larger programs
- Thus, limited usefulness for applications with state

Summary

- Managed platforms allow developers to focus more on their applications and less on operations
 - PaaS: infrastructure and runtime managed, yet users typically still allocate resources
 - FaaS: users only write and upload their functions, provider manages allocation and scaling of resources
- Platforms available for all kinds of use cases: e.g. long-running Web applications (e.g. Azure Platform), event processing (e.g. AWS Lambda), scalable data processing jobs (e.g. AWS EMR)
- New cloud execution model: Serverless Computing

References

- [1] McSherry, Frank, Michael Isard, and Derek G. Murray. "Scalability! but at what cost?." Proceedings of the 15th USENIX conference on Hot Topics in Operating Systems 2015.
- [2] P. Mell, T. Grance: "The NIST Definition of Cloud Computing", Technical Report, National Institute of Standards and Technology, 2011, <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>
- [3] I. Fried: "Inside one of the world's largest data centers", CNET, 2009, http://news.cnet.com/8301-13860_3-10371840-56.html
- [4] J. Haridas, N. Nilakantan, B. Calder: "Windows Azure Table - Programming Table Storage", 2009, <http://go.microsoft.com/fwlink/?LinkId=153401&clcid=0x409>
- [5] B. Calder: "Windows Azure Queue - Programming Queue Storage", 2008, <http://go.microsoft.com/fwlink/?LinkId=153402&clcid=0x409>
- [6] D. Chappell: "The Windows Azure Programming Model", 2010, <http://go.microsoft.com/?linkid=9751501&clcid=0x409>
- [7] Leitner, Philipp, et al. "A mixed-method empirical study of Function-as-a-Service software development in industrial practice." Journal of Systems and Software (2018)
- [8] Adzic, Gojko, and Robert Chatley. "Serverless computing: economic and architectural impact." Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering
- [9] Lee, Hyungro, Kumar Satyam, and Geoffrey Fox. "Evaluation of production serverless computing environments." 2018 IEEE 11th International Conference on Cloud Computing (CLOUD)
- [10] Wang, Liang, et al. "Peeking behind the curtains of serverless platforms." 2018 USENIX Annual Technical Conference
- [11] Roberts, Mike "Serverless Architectures", <https://martinfowler.com/articles/serverless.html>
- [12] Gojko Adzic "Designing for the Serverless Age", <https://gojko.net/2017/10/05/serverless-design-gotocph.html>
- [13] Markus Thömmes "Uncovering the magic: How serverless platforms really work!", <https://medium.com/openwhisk/uncovering-the-magic-how-serverless-platforms-really-work-3cb127b05f71>

References

- [14] Jonas, Eric, et al. "Cloud Programming Simplified: a Berkeley View on Serverless Computing", arXiv preprint arXiv:1902.03383 (2019).
- [15] Hellerstein, Joseph M., et al. "Serverless Computing: One Step Forward, Two Steps Back", *arXiv preprint arXiv:1812.03651* (2018).
- [16] Stephan Ewen, "Stream Processing and Applications in the Modern Age", Keynote at Flink Forward Berlin (2019), https://www.youtube.com/watch?v=bun0qQOX_cY.
- [17] <https://www.ververica.com/blog/announcing-stateful-functions-distributed-state-uncomplicated>, accessed January 21, 2020.